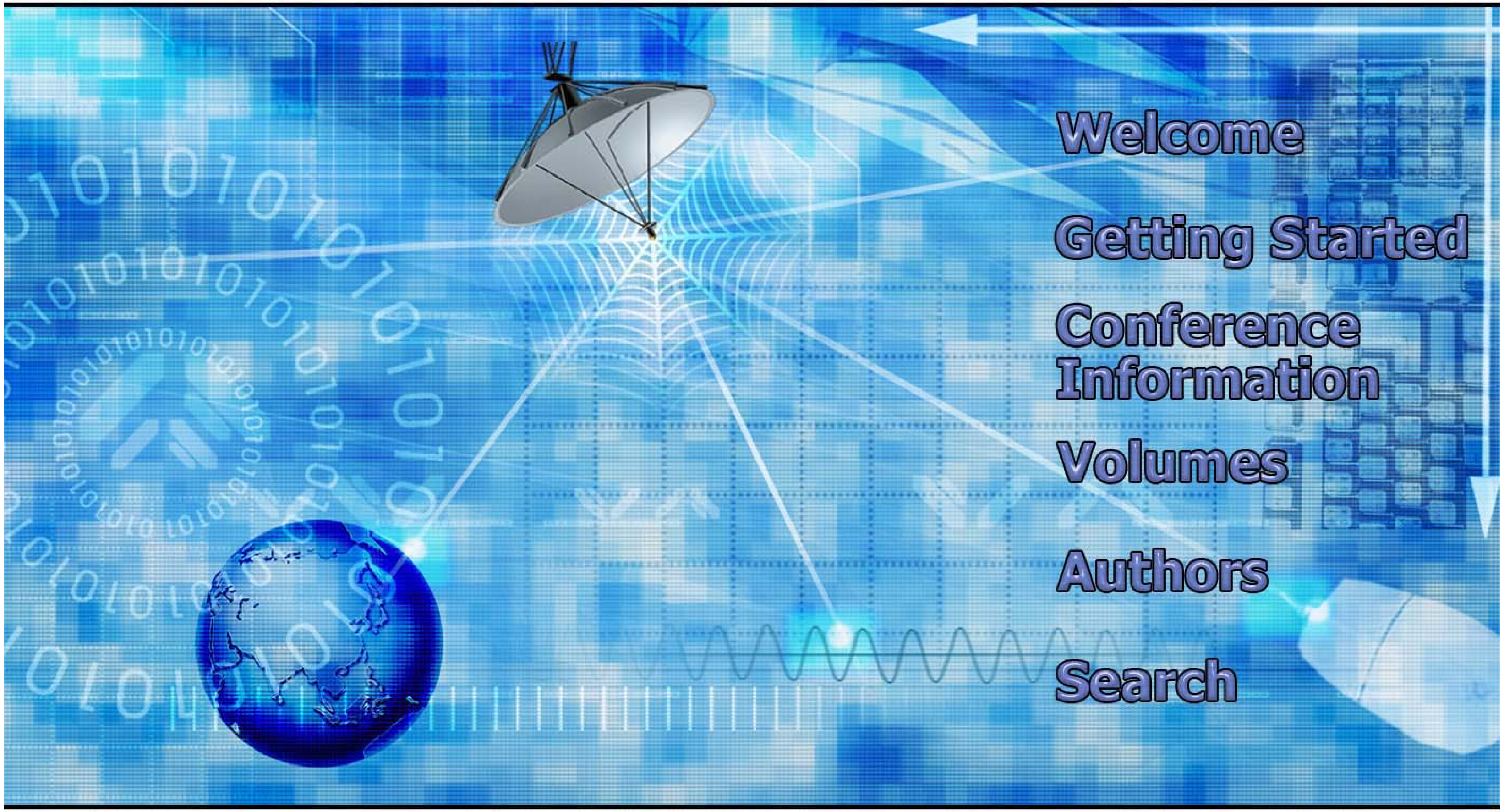
The 4th International Conference on Wireless Communications, Networking, and Mobile Computing

October 12-17, 2008 Dalian, China



IEEE WICOM 2008 IEEE COMMUNICATIONS SOCIETY

Sessions by Volume

Volume 4: Network Technologies (2)

- ❑ Sensor Networks
- ❑ Network Protocol, Traffic Analysis and Congestion Control

Click on a session for a list of papers.

Papers by Session

Network Protocol, Traffic Analysis and Congestion Control

Papers by Session

Click on a title to see the paper.

Dynamic Partial CRC with Flexible Chunk Policy for SCTP over Lossy Channel

Lin Cui
Department of Computer Science
Kyungpook National University
Daegu, Korea
cuilin@cs.knu.ac.kr

Seok J. Koh
Department of Computer Science
Kyungpook National University
Daegu, Korea
sjkoh@knu.ac.kr

Abstract—**Generally only individual regions suffer bit errors as a packet travels a lossy channel. At the receiver side, however, the entire packet has to be discarded even if only one bit is erased. This may result in degradation of the throughput performance of Stream Control Transmission Protocol (SCTP) over wireless networks with high bit error rate. This paper proposes a dynamic partial Cyclic Redundancy Check (CRC) scheme with flexible chunk policy to enhance SCTP performance. In the scheme, a packet can be partitioned into different regions based on the measured end-to-end packet corruption rate and each region can have its own individual checksum item. With the help of the checksum items, the receiver can discard the corrupted portion only and those available chunks can be recovered. Simulation results show that the performance gain of the proposed scheme could be significant in worse-case scenarios, compared to the standard SCTP.**

Keywords-SCTP; corruption; checksum; partial CRC; wireless networks

I. INTRODUCTION

The Stream Control Transmission Protocol (SCTP) [1] is originally designed for signaling messages over IP networks, and now has been extended as a general-purpose reliable transport layer protocol. Differently from Transport Control Protocol (TCP) which follows the strict byte-order delivery policy, the SCTP natively supports multiple chunks per packet and each data chunk in an association is assigned a unique transmission sequence number (TSN). That is, chunk is the information unit in SCTP, but not byte.

On the other hand, the SCTP is designed based on the window-based error and congestion control [1, 2]. As done in TCP, a corrupted packet is regarded as a lost one and thus induces the retransmission request as well as the decrease of the congestion window (*cwnd*) at the sender side. Moreover, the unwanted timeout will be incurred inevitably once the retransmitted packet suffers random bit errors. This will cause the further degradation of SCTP throughput over wireless networks with high bit error rate (BER).

Several works have been done to improve the SCTP throughput performance against corruption. The work in [3] introduces a MAC-Error-Warning (MEW) method with a new type of MAC (Medium Access Control) packet. The MEW method is similar to the Automatic Request (ARQ) mechanism, but a special MAC packet is generated at the MAC layer before the data chunk is discarded, which contains detailed information (including the stream ID and the sequence number) to notify the upper layer of the failed transmission. Arriving at the transport layer, the MEW packet will be analyzed by the SCTP source and all the useful information will be recovered so as to trigger a fast retransmission of the chunk without degrading the congestion window. The work in [4] utilizes the ECN (Explicit Congestion Notification) indications to detect the non-congestion packets loss. Therefore, a packets loss without simultaneous ECN message in the same data window will be regarded as non-congestion errors. In this case, a simple loss-recovery procedure can be executed without applying the normal congestion control mechanism.

Moreover, with the rapidly increased usage of wireless devices all over the world, the faster and cheaper link level (e.g., 802.11) mechanisms have tended to be simple (e.g., 802.11's ARQ mechanism). This trend results in a high and variable residual erasure rate (e.g., 10-50% erasure rate observed by [6]) that needs to be ultimately handled end-to-end [5].

It is noted that an SCTP segment can carry multiple chunks, but only one overall checksum is contained in its common header. On reception of an SCTP segment, if the checksum field indicates a corruption, the entire segment will be discarded no matter how many chunks are carried and whether or not all of them are corrupted. When corruption occurs only in individual chunks, however, if the receiver can recover the available ones from the corrupted segment, the sender may reduce the retransmission amount and avoid the unnecessary behaviors of both halving congestion window and unwanted timeouts. The precondition is that the sender can identify which ones are corrupted.

In this paper a dynamic partial Cyclic Redundancy Check (CRC) with flexible chunk policy is proposed to enhance the SCTP performance by dynamically adding a checksum chunk. In the scheme, once a sender affirms that its peer also supports the optional partial CRC checksum scheme (this is done by exchanging INIT/INIT-ACK chunks in the association establishment phase), the sender can dynamically partition a packet into one or more regions and withdraw/insert the partial CRC checksum chunk from/into the SCTP segments, based on the measured end-to-end packet corruption rate.

On the other hand, if the overall checksum in the common header fails at the receiver side, the SCTP receiver will check whether there is the optional checksum chunk or not. If yes, each partial CRC checksum item contained in the checksum chunk will be verified in turn. As a result, those available chunks can be recovered from the corrupted packet. Further, the associated TSNs will be reported to the sender via the subsequent SACK chunks so as to trigger the fast retransmission without shrinking the congestion window at the sender side.

The rest of this paper is organized as follows. Section II briefly represents some extended and newly defined chunks. In Section III, we describe the error and congestion controls based on the dynamic partial checksum scheme. Section IV shows some simulation results using the ns-2 networks simulator. Finally, we conclude this paper in Section V.

II. Extensions of SCTP chunks

We present the suggestion on dynamic partial CRC checksum with flexible chunk policy according to the following considerations. That is, generally only individual regions suffer bit errors as a packet travels a lossy channel. At the receiver side, however, the entire packet has to be discarded even if only one bit is erased, since the existing CRC mechanisms perform an overall checksum calculation. Nevertheless, for the environments with high BER, we may reduce the retransmission amount as much as possible if we apply dynamic partial CRC since only the corrupted portion will be discarded and those available chunks can be recovered. Thus the dynamic partial CRC mechanism may perform better than the overall checksum scheme under worse-case channel condition. For this purpose, the format of SACK chunk is extended, and a checksum chunk and an optional parameter (for both INIT and INIT-ACK chunks) are newly defined.

A. Optional parameter for both INIT and INIT-ACK Chunks

The dynamic partial CRC with flexible chunk policy is suggested in this paper as an optional choice of the standard SCTP. Therefore, in the association establishment phase, the source and the destination have to affirm whether or not to support the optional partial CRC checksum chunk over the association. This can be done by exchanging a newly defined optional parameter (e.g. "13") for this purpose, using the SCTP INIT and INIT-ACK chunks.

B. Proposed Checksum Chunk

The proposed optional partial CRC checksum chunk obeys the chunk specifications defined in RFC2960 [1]. As an example, Fig. 1 illustrates the proposed format of the chunk.

In the figure, each checksum item consists of the 4-byte checksum value field and the 2-byte coverage field, which is a total 6-byte in length. More specifically, the coverage range of each checksum item can be obtained as per the following formulas:

$$left\_edge_1 = 0 \qquad (1)$$

$$right\_edge_i = left\_edge_i + coverage\_length_i - 1 \qquad (2)$$

$$left\_edge_i = right\_edge_{i-1} + 1 \qquad (i>1) \qquad (3)$$

where i = 1, 2, …*n*, and *n* is the number of the partial checksum items contained in the checksum chunk.

With the help of the partial checksum items, the sender can dynamically adjust the chunk size and number for the pending forward packets, depending on the measured packet corruption rate. At the receiver side, the receiver can recover the most available chunks and report the corruption event to the sender. In this way, the proposed scheme can reduce the retransmission amount and avoid the significant performance degradation from the strong corruption.

C. Extended SACK Chunk

When a checksum chunk is extracted from a segment at receiver side, the receiver has to check each partial checksum item if the default overall checksum fails. Once individual corrupted chunks are picked out, the associated TSNs and timestamp need to be appended in the end of the subsequent SACK chunk.

It is noted that for each segment with single data chunk, an associated corruption item is composed of a corruption TSN and a corresponding timestamp ('explicit corruption item'), which is used to explicitly report a corrupted chunk to the sender. As opposed to the explicit corruption item, the checksum item for the corrupted segment with multiple data chunks consists of either the first erased chunk's TSN while all data chunks are corrupted or all available chunks' TSNs if any chunks are recovered ('implicit corruption item'). Therefore, on extraction of an implicit corruption item from a SACK chunk, the sender needs to infer which unacknowledged packet possesses these chunks. Of course, the other chunks in the same packet must be corrupted if the recovered chunks' number is not zero. Technically, it is easy to pick out the corrupted chunks from a multiple data chunks' segment because the sender preserves the contents of all outstanding packets till they are acknowledged.

Figure 2 shows the example format of the extended SACK chunk recommended in this paper.

III. Error and Congestion Controls

A. Data Transmission

At the beginning of data transmission phase, the sender can assemble optional checksum chunk as well as one or more data chunks into segments based on the negotiation using INIT/INIT-ACK chunks. However, each segment's structure

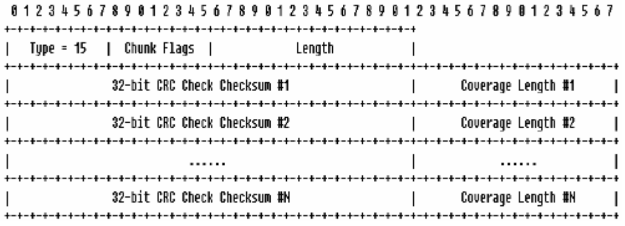
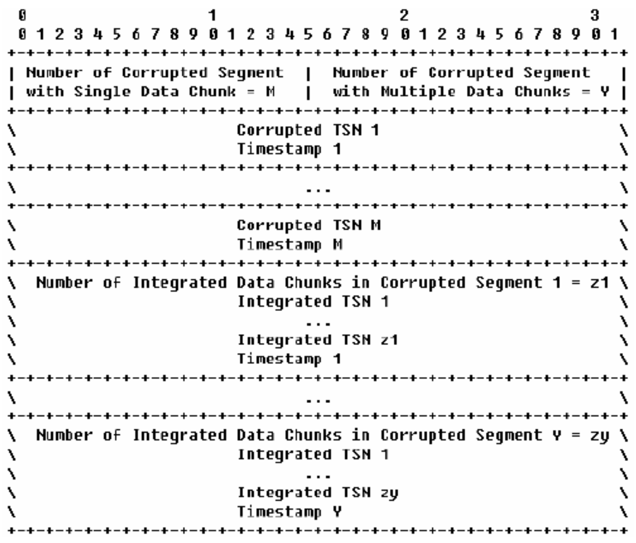


Figure 1. Format of Checksum Chunk



```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Number of Corrupted Segment  |  Number of Corrupted Segment   |
| with Single Data Chunk = M   |  with Multiple Data Chunks = Y |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                        Corrupted TSN 1                        \
\                        Timestamp 1                            \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                               ...                             \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                        Corrupted TSN M                        \
\                        Timestamp M                            \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\  Number of Integrated Data Chunks in Corrupted Segment 1 = z1 \
\                        Integrated TSN 1                       \
\                               ...                             \
\                        Integrated TSN z1                      \
\                        Timestamp 1                            \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\                               ...                             \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
\  Number of Integrated Data Chunks in Corrupted Segment Y = zy \
\                        Integrated TSN 1                       \
\                               ...                             \
\                        Integrated TSN zy                      \
\                        Timestamp Y                            \
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2. Format of the extended part of SACK chunk

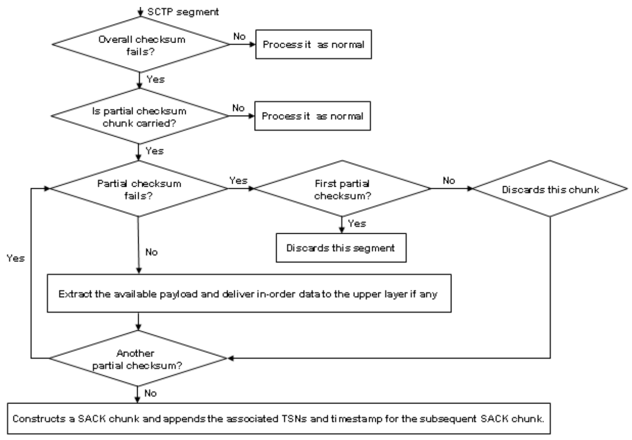

SCTP segment
Overall checksum fails?
No
Process it as normal
Yes
Is partial checksum chunk carried?
No
Process it as normal
Yes
Partial checksum fails?
Yes
First partial checksum?
No
Discards this chunk
Yes
Discards this segment
No
Yes
Extract the available payload and deliver in-order data to the upper layer if any
Another partial checksum?
No
Constructs a SACK chunk and appends the associated TSNs and timestamp for the subsequent SACK chunk.

Figure 3. Flowcharts of processing checksum chunk

might not be changeless. The sender can adjust the chunk size and the number depending on the measured packet's corruption rate. For example, when the packet corruption rate decreases down to a lower threshold below, the checksum chunk can be withdrawn and one data chunk's payload is recommended; whereas if the packet corruption rate exceeds a higher threshold, both checksum chunk and multiple data chunks' payload are preferred.

B. *Detection of Corruption Events*

On reception of an SCTP segment, the receiver will first verify the integrity of the whole SCTP segment by checking the overall checksum of the common header. In case that the segment is corrupted, the receiver will then verify each partial checksum in turn if the checksum chunk is carried.

In particular, once checking a partial checksum fails, if it is the first one, the receiver has to discard the whole segment since the header portion may contain some wrong information. On the other hand, if the first partial checksum is proven to be valid, the subsequent partial checksums, if any, will be checked in sequence. For each of the subsequent partial checksums (only for the segment with multiple data chunks), if it is proven to be valid, the corresponding data chunk can be recovered from the corrupted segment. The detailed processing procedure of data packet is illustrated in Figure 3.

C. *Generation of SACK Chunk for Corruption*

After checking all of partial checksums, the SCTP receiver will construct a subsequent SACK chunk immediately, in which the associated TSNs and timestamp will be appended in the end as shown in Figure 2. In particular, if a single data chunk's segment is corrupted, an explicit corruption item will be appended; otherwise if the corrupted segment contains multiple data chunks, an implicit corruption item is needed. Wherein, the timestamp indicates when the corruption is detected at the receiver side. Therefore, even for the same TSN, a renewed timestamp implies a new corruption event, and thus requests another prompt retransmission. This allows the sender can retransmit the same chunk for multiple times before the retransmission timer expires. Consequently, it can let the sender avoid the unwanted timeouts which might be induced in conventional SCTP in case that the retransmitted chunks suffer random bit errors in wireless channel.

D. *Error and Congestion Controls*

In the standard SCTP, both lost and corrupted chunks are retransmitted by either the timer-based retransmission or the fast retransmission. In the proposed scheme, however, the corrupted chunk can be processed differently from the lost one since the corruption itself indicates an explicit retransmission request.

By comparing the history records with the corruption items enclosed in the end of a SACK chunk, the sender can easily infer whether a corruption item is to report a new corruption event. If the sender determines that a corruption item reports a new corruption, it will retransmit the corrupted chunks immediately without deflating its congestion window.

Figure 4 shows the flowchart of processing a SACK chunk

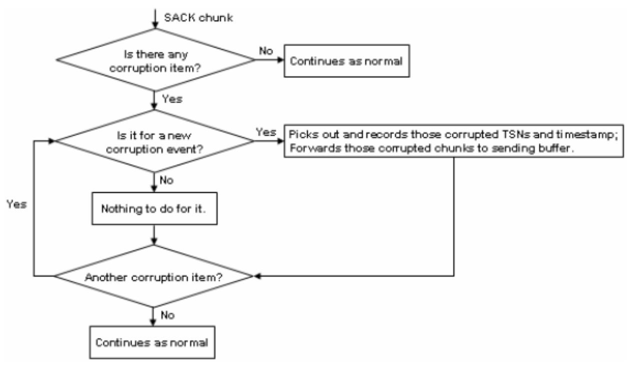

SACK chunk
Is there any corruption item?
No
Continues as normal
Yes
Is it for a new corruption event?
Yes
Picks out and records those corrupted TSNs and timestamp; Forwards those corrupted chunks to sending buffer.
No
Nothing to do for it.
Yes
Another corruption item?
No
Continues as normal

Figure 4. Flowchart of processing SACK chunk

by the sender. In the figure, when the sender receives a SACK chunk, it first checks whether there are any corruption items. If no, the sender processes it as normal. If any, the sender further determines whether the first item reports a new corruption event. If it does, then the sender records the corrupted TSNs and timestamp for prompt retransmission. Otherwise the sender simply ignores it and continues to check the next one.

By tracing the measured packet corruption rate, the sender can adjust its chunk policy in time. For example, a packet can carry one or more data chunks depending on the different corruption levels, or the sender can withdraw the partial checksum chunk from the data chunks if the corruption rate is teeny. All of the behaviors are simply and repeatable since the partial CRC checksum chunk is proposed as an option of the standard SCTP. In this way, the proposed scheme can significantly improve the throughput performance of SCTP over wireless network with a high BER.

IV. SIMULATION RESULTS

We evaluate the proposed scheme using the ns-2 network simulator (version 2.30) [7] with the simple test topology, in which two endpoints communicate directly through a single path that has the bandwidth of 2 Mbps and the transmission delay of 35ms. In each experiment, we perform the file transfer application over SCTP for 200 seconds and compare the goodputs (Kbps) of the standard SCTP with those of the proposed scheme in the cases that every packet carries 1, 2, 4, 6, 8, 10 data chunks, respectively.

To emulate packet corruptions, a two-state error model is modified to add a corruption flag in every corrupted packet's header instead of dropping it.

In the model, two states of 'good' and 'bad' are expressed in terms of average error rates λ and β, transition probabilities p and q, and average 'good' period of $t1$ seconds and 'bad' period of $t2$ seconds. If the link is in a 'good' state at present, it will continue to stay in the 'good' state with probability p, or transfer to the 'bad' state with a probability 1-p at the next instance. Also, if the link is in a 'bad' state at the current instance, then it will continue to stay in the 'bad' state with probability q, or transfer to the 'good' state with a probability 1-q at the next instance. When the link is in the 'bad' state, a SCTP segment experiences a packet corruption in the network with the probability β. Table I summarizes the parameter values used for the error model.

Table I. Parameters used for two-state error model

State	*Average Period*	*Transition Probability*	*Packet corruption Rate*
Good	t1=0.5 seconds	p=0.5, (1-p)=0.5	λ= 0
Bad	t2=0.5 seconds	(1-q)=0.5, q=0.5	β =0.1~50%

Table II. Packet structure and drop rates

Packet drop rate	Packet Structure			
	Data Chunk Number	*Checksum Chunk Size*	*Data Chunk Size*	*Packet Size*
58 β /1498	1	10	1456	1498
70 β /1494	2	22	720	1494
82 β /1490	4	34	356	1490
94 β /1494	6	46	236	1494
106 β /1498	8	58	176	1498
118 β /1462	10	70	136	1462

Notice that in the experiments, the standard SCTP uses the fixed-size data chunk (1468 bytes) and each packet carries only one data chunk. Thus every packet has a fixed 1500-byte in size. Moreover, since the standard SCTP regards the corruption as loss, all corrupted packets are forcedly dropped by the receiver. Therefore, the packet drop rate is equal to β in bad states for the standard SCTP.

On the contrary, the proposed scheme performs the experiments using various-size data chunks. The detailed packet structure information is shown in Table II. Furthermore, in order to emulate the scenarios where bit errors occur in the header portion of SCTP packets, the packet drop rate of bad state is set to the proportion of the header size over the packet size. In this paper, the header scope covers IP header, SCTP common header, checksum chunk and the first data chunk's header. When header is corrupted, all data chunks cannot be recovered. The detailed packet drop rate is also given in Table II. Also, we assume in this paper that only one data chunk suffers bit errors in every corrupted packet.

The simulation results are shown in Fig. 5. In the figure, the horizontal axis denotes the packet corruption rate ranged from 0.1% to 50% (that is β value in bad state of the two-state error model) and vertical axis indicates the goodput that is calculated based on the Cumulative TSN of the final SACK chunk.

From Fig. 5 we can see that the standard SCTP gives the higher goodputs over the proposed scheme when the packet corruption rate (β value) is smaller than 1.5% roughly. This is

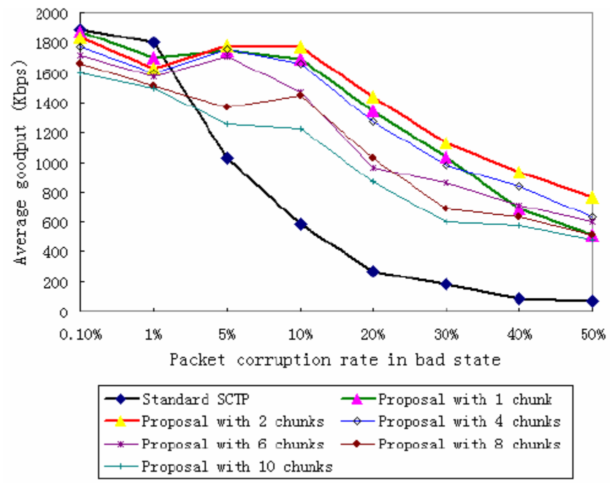


Figure 5. Goodputs with different β values

because when β is smaller, the proposed scheme keeps a large transmission window unchanged since there is no corruption occurs in packet header. This will sometimes incur the receiving buffer blocking problem at receiver side.

Beyond that point, the proposed scheme starts to appear its advantages and outperforms the standard SCTP. Such performance gain is anticipated since the proposed scheme can recover most available payload form the corrupted packets and can decrease its *cwnd* only when corruption occurs in packet header portion. Whereas the standard SCTP interprets all corruption events as the indication of network congestion in the same way and blindly halves its *cwnd* repeatedly.

However, through comparing the different behaviors of the checksum chunk with various-size data chunks, we find that too many data chunks (e.g., more than three) may result in some undesirable side effects, such as 1) overhead caused by too huge checksum chunk as well as too many chunk headers, 2) much serious receiving buffer blocking problems and 3) raised packet loss rate incurred by header corruption. This will waste the more available bandwidth and decrease the transmission efficiency.

V. CONCLUSIONS

In this paper, we propose a dynamic partial CRC checksum chunk with flexible chunk policy to enhance the SCTP throughput performance under the environments with high BER. From the simulation results, we can see that applying the proposed checksum chunk could provide a significant throughput performance gain over the standard SCTP when the packet corruption rate is high.

The performance gain of the proposed scheme comes from the following features: Firstly, the proposed scheme can recover the available data chunks from the corrupted packet so as to reduce the retransmission amount as much as possible. Secondly, by comparing the history records with the corrupted TSN as well as the corresponding timestamp enclosed in the end of SACK chunk, the proposed scheme can exploit a robust retransmission policy to avoid the unwanted timeouts which might be induced in conventional SCTP. Thirdly, the proposed scheme can distinguish the chunk corruptions from the chunk losses by using additional checksum chunk. Hence, it can avoid unnecessary deflation of the congestion window in the face of packet corruption.

ACKNOWLEDGMENT

This research was supported by the MKE (Ministry of Knowledge Economy) of Korea, under the ITRC support program supervised by the IITA (IITA-2008-C1090-0801-0026).

REFERENCES

[1] R. Stewart et al., “Stream control transmission protocol,” IETF, RFC 2960, Oct. 2000.

[2] M. Allman et al., “TCP congestion control,” IETF, RFC 2581, Apr. 1999.

[3] D. Wang; S. Yang; W. Sun; "A Mac-Error-Warning Method for SCTP Congestion Control over High BER Wireless Network", Wireless Communications, Networking and Mobile Computing, 2005. Proceedings. 2005 International Conference on Volume 1, Sept. 23-26, 2005 Page(s):513 - 516.

[4] G. Ye; S., T.N.; M. J Lee; "Improving Stream Control Transmission Protocol Performance Over Wireless networks", Selected Areas in Communications, IEEE Journal on Volume 22, Issue 4, May 2004 Page(s):727 – 736.

[5] O. Tickoo, V. Subramanian, S. Kalyanaraman and K. K. Ramakrishnan "LT-TCP: End-to-End Framework to Improve TCP Performance over Networks with Lossy Channels," In proceedings of IEEE 13th International Workshop on Quality of service (IWQoS), Passau, Jun 21-23, 2005.

[6] D. Aguayo, J. Bicket, S. Biswas, G. Judd and R. Morris, ”Link-level Measurements from an 802.11b Mesh Network”,SIGCOMM 2004, Aug 2004.

[7] Network Simulator (ns-2), available from http://www.isi.edu/nsnam/ns/